PROGRAMACIÓN II – UNIDAD 1
TEORÍA POO

Ing. Gastón Weingand (gaston.weingand@uai.edu.ar)

Agenda

1. Repaso de estructuras fundamentales
2. Revisión de conceptos POO

Agenda

1. Repaso de estructuras fundamentales
2. Revisión de conceptos POO

1-Sentencias condicionales e iteradores

Condicionales simples

Ejercicio 3. Definir, usando condicionales, la función maximo tal que maximo(x,y) es el máximo de x e y. Por ejemplo,

```
>>> maximo(2, 5)
5
>>> maximo(2, 1)
2
```

1-Sentencias condicionales e iteradores

```
In [17]: def maximo(x, y) :
           if x > y:
             return x
           else:
             return y

In [18]: maximo(2, 5)

Out[18]: 5

In [19]: maximo (2, 1)

Out[19]: 2
```

1-Sentencias condicionales e iteradores

Condicionales múltiples

Ejercicio 4. Definir la función signo tal que signo(x) es el signo de x. Por ejemplo,

```
>>> signo(5)
1
>>> signo(-7)
- 1
>>> signo(0)
0
```

1-Sentencias condicionales e iteradores

In [20]:
```
def signo(x):
  if x > 0:
    return 1
  elif x < 0:
    return -1
  else:
    return 0
```

In [21]:
```
signo(5)
```

Out[21]: 1

In [22]:
```
signo(-7)
```

Out[22]: -1

In [23]:
```
signo(0)
```

Out[23]: 0

1-Sentencias condicionales e iteradores

También existe una construcción similar al operador ? de otros lenguajes, que no es más que una forma compacta de expresar un `if else`. En esta construcción se evalúa el predicado C y se devuelve A si se cumple o B si no se cumple: *A* `if` *C* `else` *B*. Veamos un ejemplo:

```
var = "par" if (num % 2 == 0) else "impar"
```

1-Sentencias condicionales e iteradores

Sentencia for

```
word = input("Introduce una palabra: ")
vocals = ['a', 'e', 'i', 'o', 'u']
for vocal in vocals:
    times = 0
    for letter in word:
        if letter == vocal:
            times += 1
    print("La vocal " + vocal + " aparece " + str(times) + " veces")
```

1-Sentencias condicionales e iteradores

Sentencia for range (start (opcional) = 0, stop, step (opcional) = 1)

```
subjects = ["Matemáticas", "Física", "Química", "Historia", "Lengua"]
scores = []

for subject in subjects:
    score = input("¿Qué nota has sacado en " + subject + "?")
    scores.append(score)
for i in range(len(subjects)):
    print("En " + subjects[i] + " has sacado " + scores[i])
```

1-Sentencias condicionales e iteradores

Sentencia while

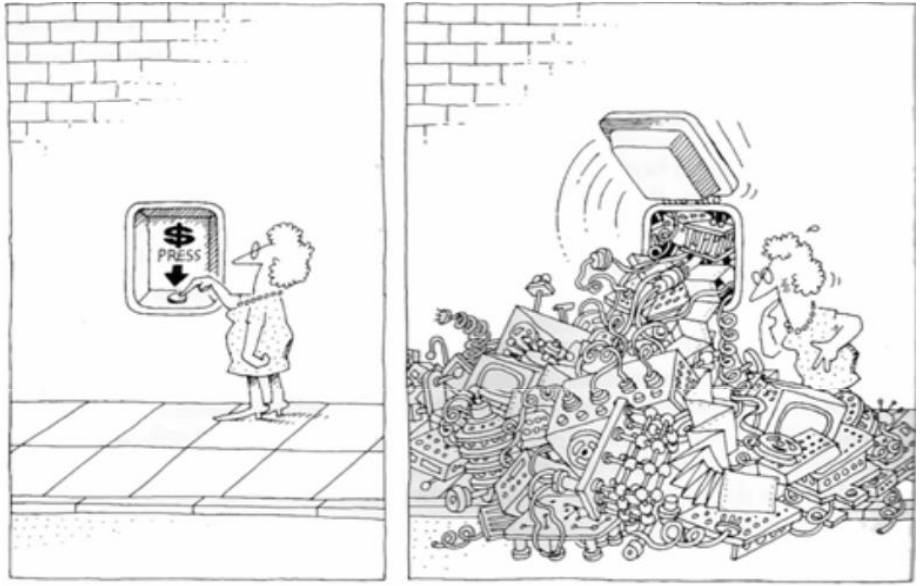
```
while True:
    entrada = input("Ingrese un mensaje de esperanza...")
    if entrada == "adios":
        break
    else:
        print (entrada)
```

Agenda

1. Repaso de estructuras fundamentales
2. Revisión de conceptos POO

Paradigma Orientado a objetos

La misión del equipo de desarrollo de software es producir ilusión de simplicidad.



Lenguajes Orientados a Objetos

- Simula
- Smalltalk
- C++
- Object Pascal
- Java
- C#
- Python

Paradigmas de Objetos

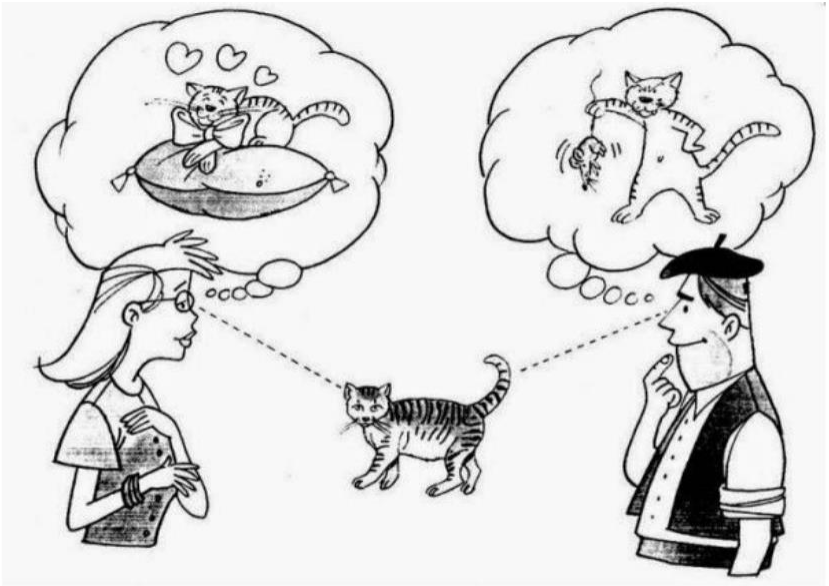
- La principal abstracción de nuestro nuevo paradigma son los objetos, que representan distintos tipos de entidades.

Paradigmas de Objetos

¿QUE ES UN OBJETO?

Es algo que puedo representar a través de una idea, un concepto.
Tiene entidad.
Por ejemplo, un gato es algo del mundo real, que podemos conceptualizar fácilmente.

Paradigmas de Objetos



Representaciones mentales de un objeto real

Paradigmas de Objetos

El modelo mental es una simplificación del objeto real.
Dos personas pueden ver la misma cosa, sin embargo están abstrayendo representaciones diferentes porque tienen objetivos distintos.

Lo que importa para el paradigma no es el objeto real, sino nuestro modelo mental.

Paradigmas de Objetos

Todo lo que existe en mi cabeza puede ser tomado como un objeto.

Un sistema en el paradigma orientado a objetos es :

> “Un sistema es un conjunto de objetos que se envían mensajes para alcanzar un determinado objetivo”

Paradigmas de Objetos

Un programa basado en este paradigma es un conjunto de objetos que interactúan entre sí en un ambiente mandándose mensajes para lograr un objetivo determinado.

Conceptos de POO

¿Qué es una clase?

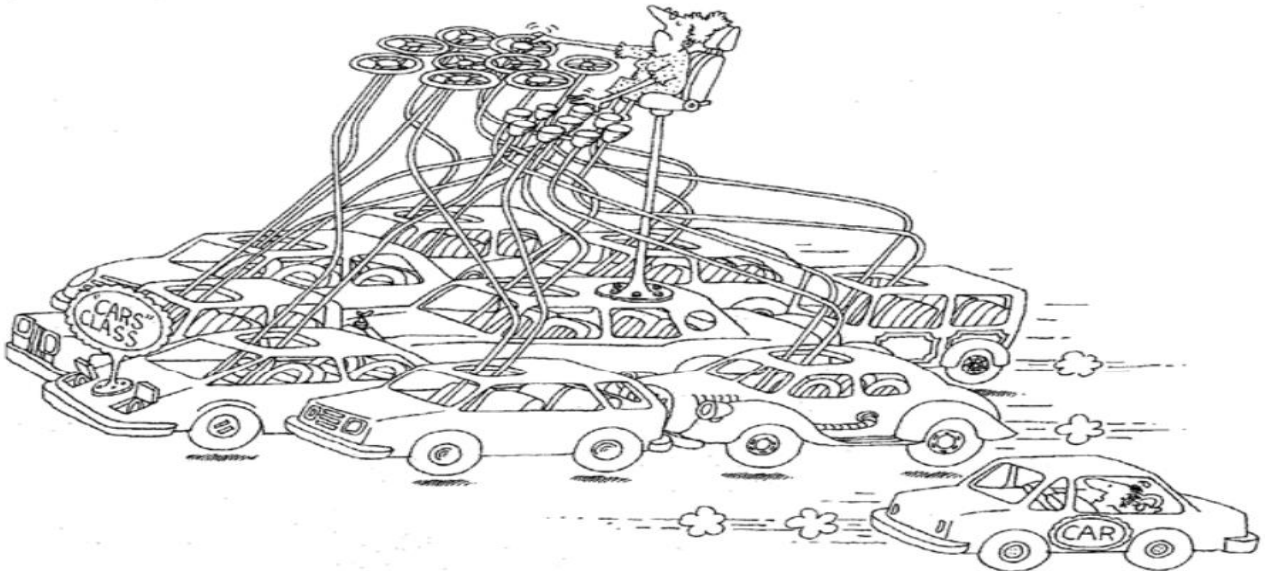
Una clase puede definirse como la agrupación o colección de objetos que comparten una estructura común y un comportamiento común.

Consta de atributos y métodos.

Todo objeto pertenece a una clase.

Un objeto es una instancia de una clase.

Conceptos de POO



Conceptos de POO

Una clase es un nivel de abstracción alto: permite describir un conjunto de características comunes para los objetos que representa.

Sintaxis algorítmica es:

Clase <nombre de la clase>

Las clases son descripciones estáticas, durante la ejecución de un programa sólo existen los objetos, no las clases.

Conceptos de POO

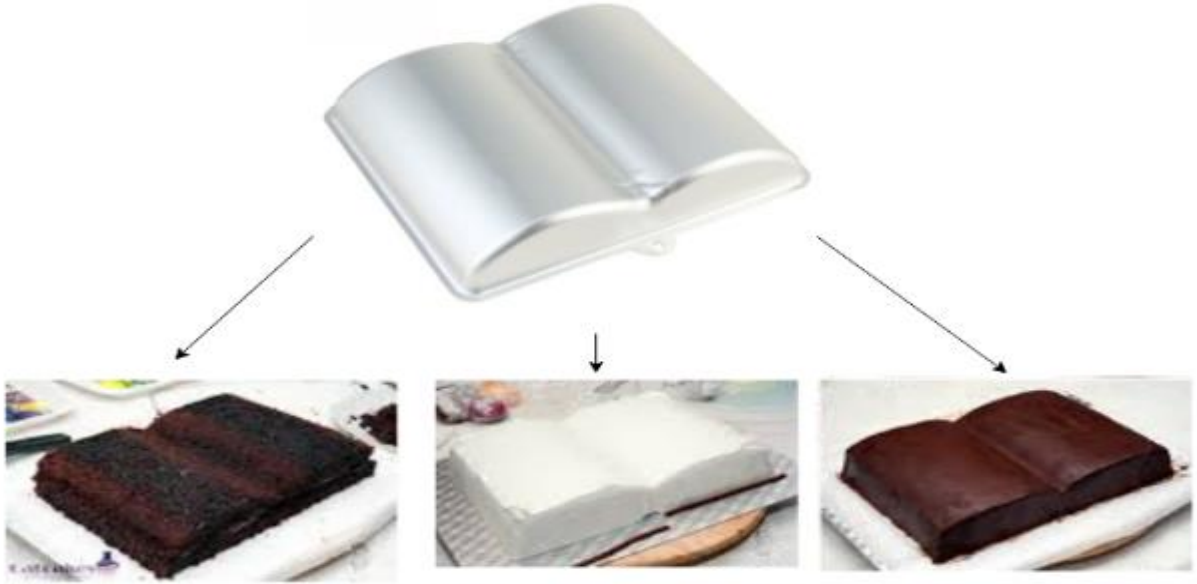
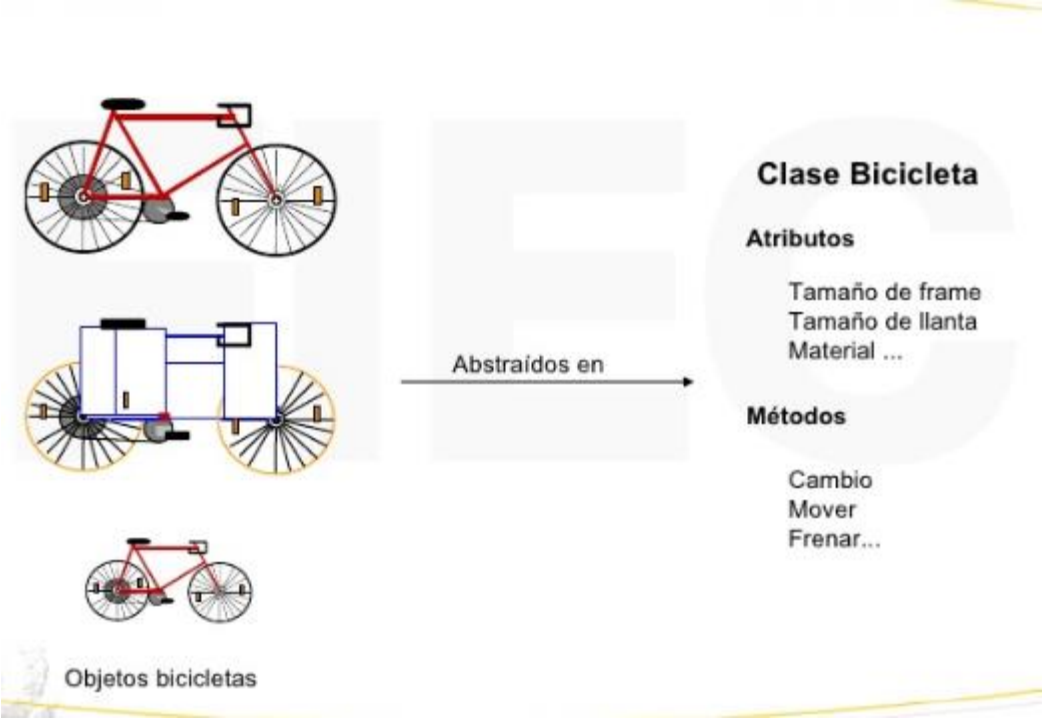


Fig. 2: con el *molde* (hueco) instanciamos una chocotorta, una torta glaseada y una bombón

Conceptos de POO

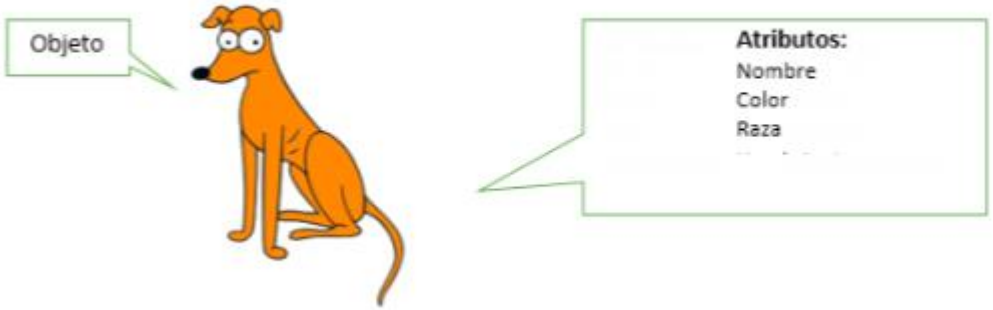


Conceptos de POO

Atributos: los datos contenidos dentro de un objeto, podemos definirlos como las características o propiedades que posee un objeto.

Los atributos se implementan como variables, constantes o estructuras de datos, y cada objeto en particular puede tener valores distintos para estas.

Conceptos de POO



Conceptos de POO

Mediante los atributos se define información oculta dentro de un objeto, la cual es manipulada solamente por los métodos definidos sobre dicho objeto.

Un atributo consta de un nombre y un valor.

Su sintaxis algorítmica es:

<Modo de Acceso> <Tipo de Dato> <Nombre del atributo>

Conceptos de POO

Los modos de acceso pueden ser:

- Publico: atributos que son accesibles fuera de la clase. Se puede representar con el símbolo +.
- Privado: atributos que son solo accesibles dentro de la implementación de clase. Se puede representar con el símbolo -.
- Protegido: atributos o métodos que son accesibles para la propia clase y sus clases hijas (subclases). Se representa con #.

Conceptos de POO

ESPECIFICADOR	*CLASE*	*SUBCLASE*	*PAQUETE*	*MUNDO*
Private	X			
Protected	X	X*	X	
Public	X	X	X	X

Conceptos de POO

Estado: el estado de un objeto esta determinado por los valores que poseen sus atributos en un momento dado.

El valor concreto de una propiedad de un objeto se llama estado.

Mensaje

¿Cómo interactúan los objetos entre sí?
A través de los mensajes.

¿Cómo envío un mensaje a otro objeto?
Primero lo tengo que conocer cuando lo conozco, sé qué mensajes le puedo mandar

¿se cómo hace para resolver mi mensaje?
No, solo lo invoco y el lo resuelve.

Mensaje

pepita es un pájaro.
Mensajes → cosas que le puedo pedir al objeto pepita que haga:

- energía (¿cuál es tu energía?)
- volá
- comé

¿Qué hace volá y comé? Depende de donde lo miro:
Si se analiza desde el que pide a pepita que vuele y que coma. No se exactamente lo que hace, pero se que quiero que vuele y que coma. No me interesa cómo está implementado.

Mensaje

Ese beneficio se llama **abstracción**, es decir concentrarnos sólo en lo que queremos resolver y dejar los detalles que no son esenciales de lado, eso nos permite mantener la complejidad lo suficientemente acotada.

Pero…
Si soy pepita, sí sé cómo volar. Es decir se la definición interna de volar.
Desde el lado de pepita, es decir del objeto que es invocado estoy encapsulando: agrupando funcionalidades propias.

Mensaje y Método

Mensaje es lo que el objeto emisor le envía como orden al receptor.
El emisor no sabe cómo se resuelve el mensaje, sólo pide que se resuelva.

Un **mensaje** es algo que le puedo decir a un objeto.

Conceptos de POO

Mensaje: es la petición de un objeto a otro para solicitar la ejecución de alguno de sus métodos o para obtener el valor de un atributo público.

Un mensaje consta de tres partes:

- Identidad del receptor: nombre del objeto que contiene el método a ejecutar
- Nombre del método a ejecutar: solo los métodos declarados públicos.
- Lista de parámetros que recibe el método

Mensaje y Método

Un **método** es una secuencia de líneas de código que tiene un nombre.
Cuando se le envía un mensaje a un objeto, se activa un método cuyo nombre coincide con el mensaje enviado.

Ejemplo: Si tengo un objeto referenciado por la variable pepe, y pongo

pepe direccion

Entonces…

- estoy enviando el mensaje *direccion*
- se va a activar un método llamado *direccion*.

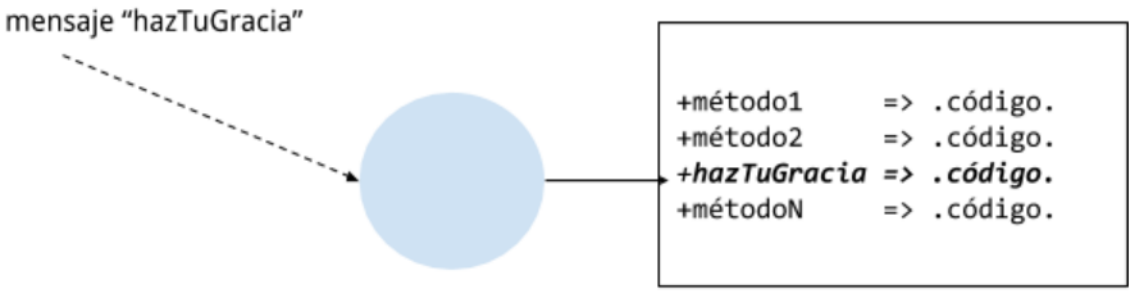
El receptor recibe el mensaje y se ejecuta un **método** (porción de código).

Mensaje y Método

La estrategia que utilizan los lenguajes para resolver dónde está el código de un método al enviar un mensaje recibe el nombre de **method lookup**.

El código a ejecutar de un mensaje se busca en un método del objeto receptor de dicho mensaje.

Mensaje y Método



method lookup básico
se busca en las definiciones del objeto un método “hazTuGracia”

Conceptos de POO

Método: son las operaciones (acciones o funciones) que se aplican sobre los objetos y que permiten crearlos, cambiar su estado o consultar el valor de sus atributos.

Cada método tiene:

- Nombre
- Cero o mas parámetros de entrada o de salida
- Algoritmo con el desarrollo del mismo

Los métodos se ejecutan cuando el objeto recibe un mensaje.

Conceptos de POO

Método Constructor: el objetivo de un constructor es el de inicializar un objeto cuando este es creado.

Los atributos de un objeto toman valores iniciales dados por el constructor.

Por convención el método constructor tiene el mismo nombre de la clase y no se le asocia un modo de acceso (es publico).

Se puede tener más de un constructor en una clase.

Conceptos de POO

```
class Coche:


    def __init__(self, gasolina):  # Constructor
        self.__gasolina = gasolina
        print("Tenemos", gasolina, "litros")
```

Conceptos de POO

Constructor por defecto: es un constructor sin parámetros que no hace nada.

Será invocado cada vez que se construya un objeto sin especificar ningún argumento, en este caso el objeto será iniciado con valor predeterminados por el sistema. Por ejemplo en Java: los String se inicializan en null, los int en 0.

Conceptos de POO

```
class Coche:


    def __init__(self):  # Constructor
        print(“Constructor por defecto")
```

Conceptos de POO

Método Destructor: los objetos que ya no son utilizados en un programa, ocupan inútilmente espacio de memoria, que es conveniente recuperar en un momento dado. Según el lenguaje de programación utilizada esta tarea es dada al programador o es tratada automáticamente por el procesador.

```
def __del__(self):
# body of destructor
```

Conceptos de POO (Mensajes)

Su sintaxis algorítmica es:

<variable_objeto>.<nombre_método>([<Lista de parámetros]);

El emisor no se entera de como se resuelve el mensaje.

El receptor recibe el mensaje y se ejecuta un método.

Ambiente

Tenemos observador y objeto:

Ambiente

Pero **¿dónde viven esos objetos?**
Necesitaría un lugar, es decir un **ambiente**: el lugar donde viven los objetos.

Ambiente

Ambiente

La palabra ambiente es conocida en otras tecnologías como

imagen, en Smalltalk
virtual machine, para Java
.net framework, para tecnología Microsoft
execution environment, para python

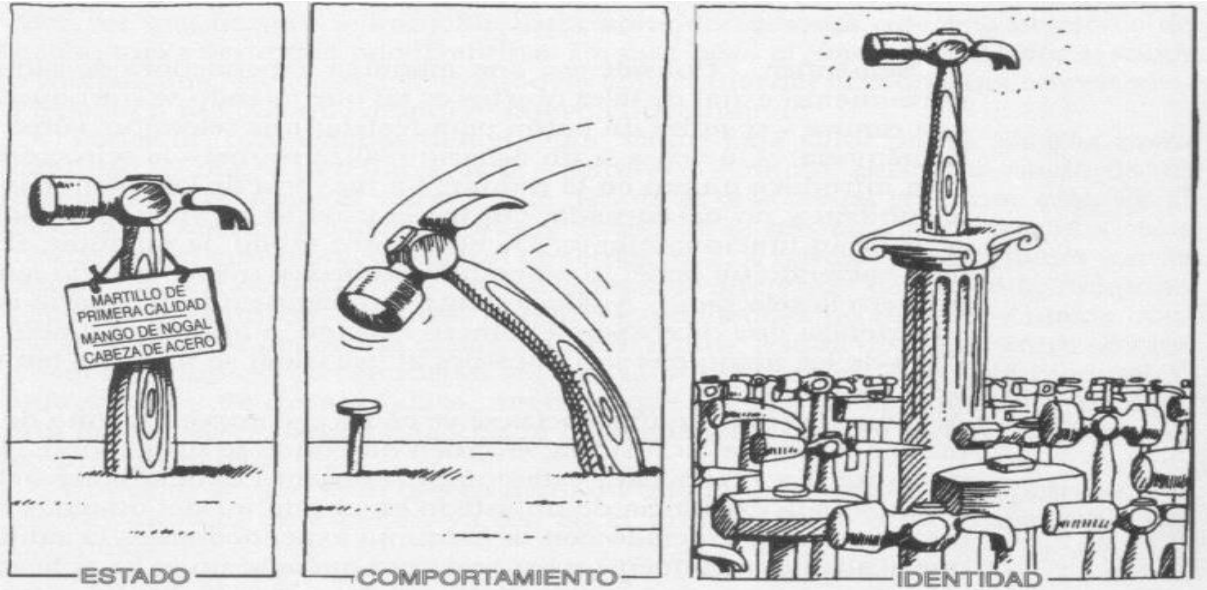
Estado de un Objeto

El estado de un objeto abarca todas las propiedades del objeto, y los valores actuales de cada una de esas propiedades.
Las propiedades de los objetos suelen ser estáticas, mientras los valores que toman estas propiedades cambian con el tiempo.

Estado de un Objeto

Un objeto puede enviar mensajes a otros objetos, para lo cual los tiene que conocer.
El conjunto de referencias que tiene un objeto representa su **estado**.

Conceptos de POO



Un objeto tiene estado, exhibe algún comportamiento bien definido, tiene una identidad única.

Conceptos de POO

Comportamiento: esta definido por los procedimientos o métodos con que puede operar dicho objeto, es decir, que operaciones se pueden realizar con él.

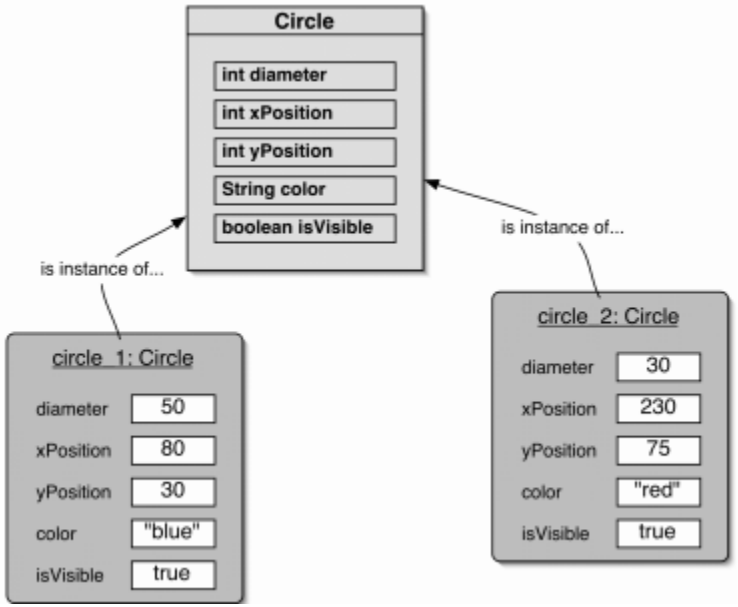
Los métodos se utilizarán para obtener o cambiar el estado de los objetos, así como proporcionar un medio de comunicación entre objetos.

Conceptos de POO

Identidad: cada objeto tiene una identidad única, incluso si su estado es idéntico al de otro objeto.

Cada objeto es único por más que haya otro con iguales atributos.

Conceptos de POO



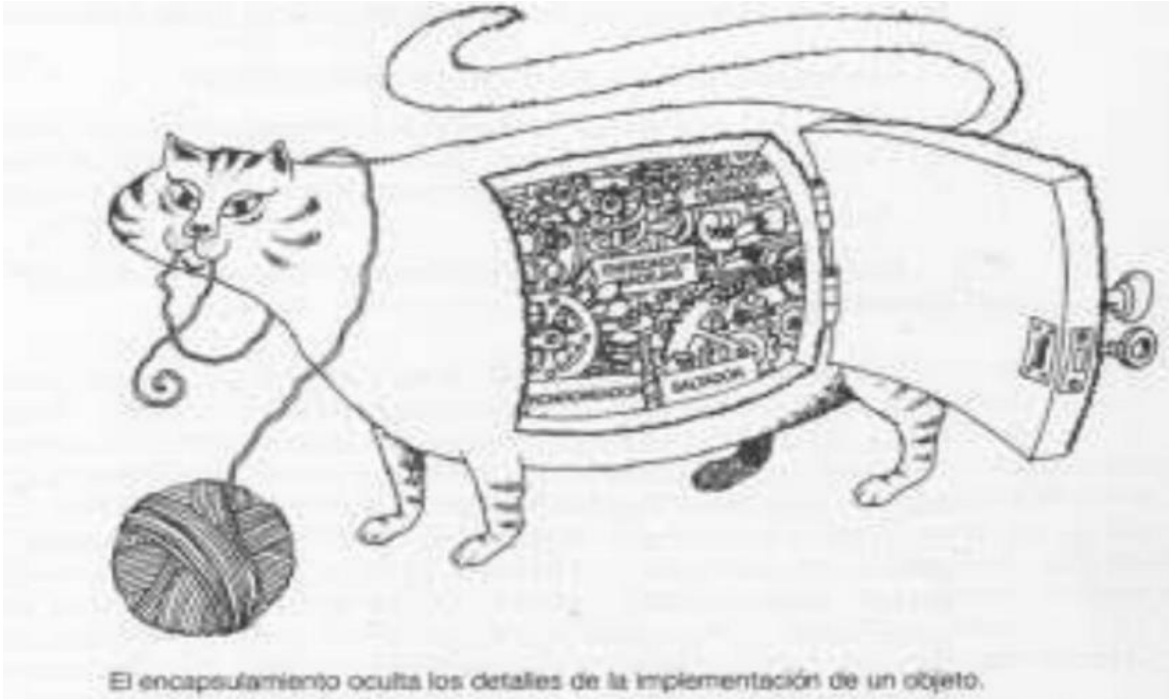
Conceptos de POO

Conceptos de POO

Encapsulamiento: Es el ocultamiento del estado de los datos miembro de un objeto de manera que solo se pueda cambiar mediante las operaciones definidas para ese objeto.

Solo se conoce el comportamiento pero no los detalles internos.

Conceptos de POO



El encapsulamiento oculta los detalles de la implementación de un objeto.

Conceptos de POO

Interfaz: es lo que un objeto publica para que otro objeto lo use.

Implementación: es lo que un objeto encapsula para definir como se termina resolviendo el mensaje.

Paradigmas de Objetos

¿Qué es lo que más nos importa de un objeto?

Lo que más nos importa son los mensajes que la puedo enviar, estos forman la **interfaz.**

Ejemplo: el gato de la imagen es Félix, que sabe maullar, comer y dormir. Veamos su interfaz

Diagrama de clases

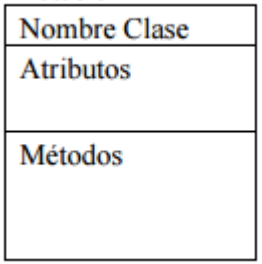
La representación gráfica de una o varias clases y sus relaciones.

Para los diagramas de clase se utilizará la notación que provee el Lenguaje de Modelación Unificado (UML).

Las clases se denotan como rectángulos divididos en tres partes: nombre de la clase, atributos y métodos.

Diagrama de clases

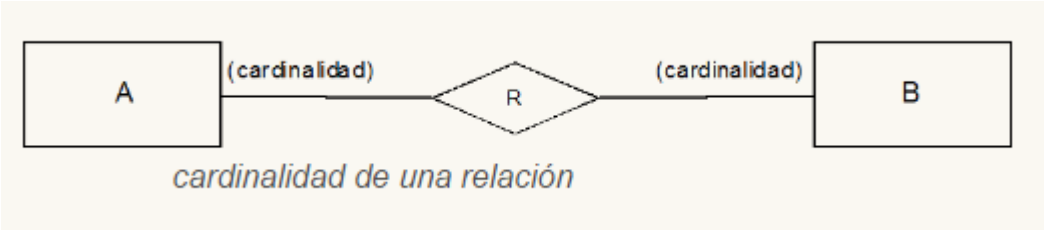
Notación:



Cardinalidad o multiplicidad

Número de ocurrencias que se pueden dar en una relación; con cuantas ocurrencias de B se puede relacionar A y con cuantas ocurrencias de A se puede relacionar B.

Cardinalidad o multiplicidad



cardinalidad de una relación

Cardinalidad o multiplicidad

Ejemplo:

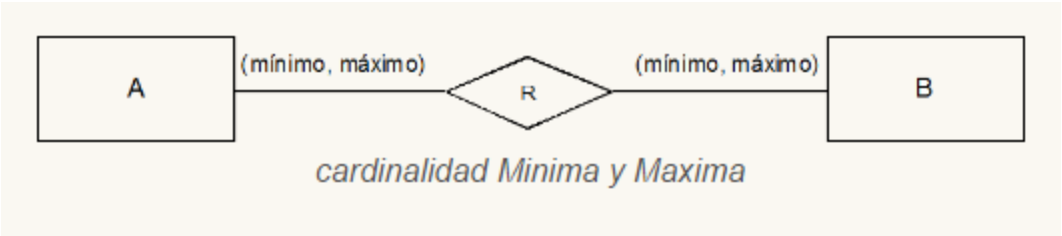
- Una persona puede comprar muchos autos y un auto es comprado por una sola persona

Se lee de izquierda a derecha y luego de derecha a izquierda (o al revés)

Cardinalidad o multiplicidad

Cardinalidad máxima: representa el número máximo de ocurrencias de una entidad con las que se puede relacionar otra ocurrencia de entidad.

Cardinalidad mínima: representa el número mínimo de ocurrencias de una entidad con las que se puede relacionar esa entidad.

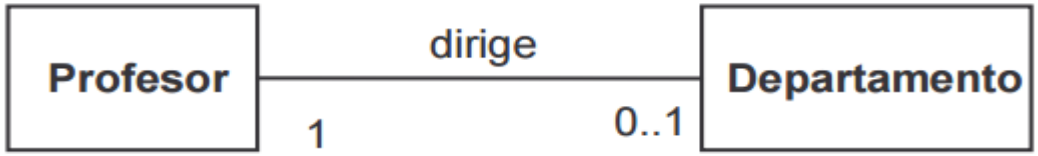


cardinalidad Minima y Maxima

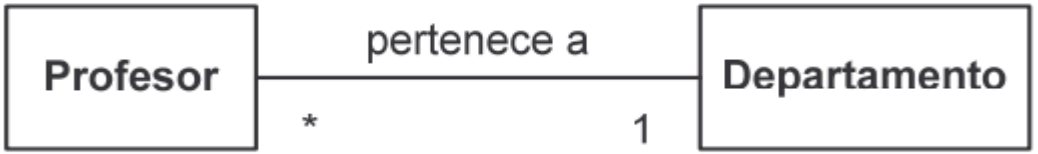
Cardinalidad o multiplicidad

Multiplicidad	Significado
1	Uno y sólo uno
0..1	Cero o uno
N..M	Desde N hasta M
*	Cero o varios
0..*	Cero o varios
1..*	Uno o varios (al menos uno)

Cardinalidad o multiplicidad



Todo departamento tiene un director.
Un profesor puede dirigir un departamento.



Todo profesor pertenece a un departamento.
A un departamento pueden pertenecer varios profesores.

Relaciones entre clase

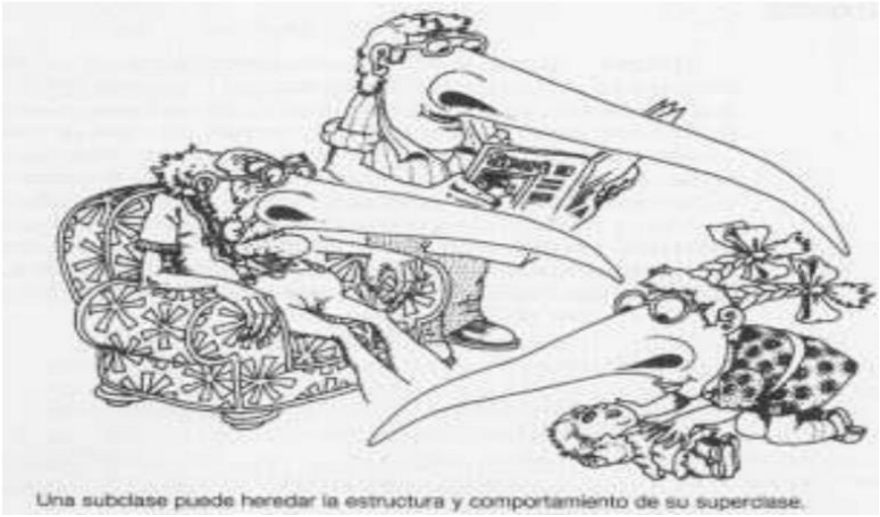
Las clases se relacionan entre sí, de manera que puedan compartir atributos y métodos sin necesidad de reescribirlos.

Relaciones entre clase

Herencia: una clase nueva se crea a partir de una existente.

Mediante la herencia las instancias de una clase hija (o subclase) pueden acceder tanto a los atributos como a los métodos públicos y protegidos de la clase padre (o superclase). Se suele hacer referencia a la relación como “es una”.

Relaciones entre clase



Una subclase puede heredar la estructura y comportamiento de su superclase.

Relaciones entre clase

Cada subclase o clase hija en la jerarquía es siempre una extensión de la clase padre, además puede incorporar atributos y métodos propios.

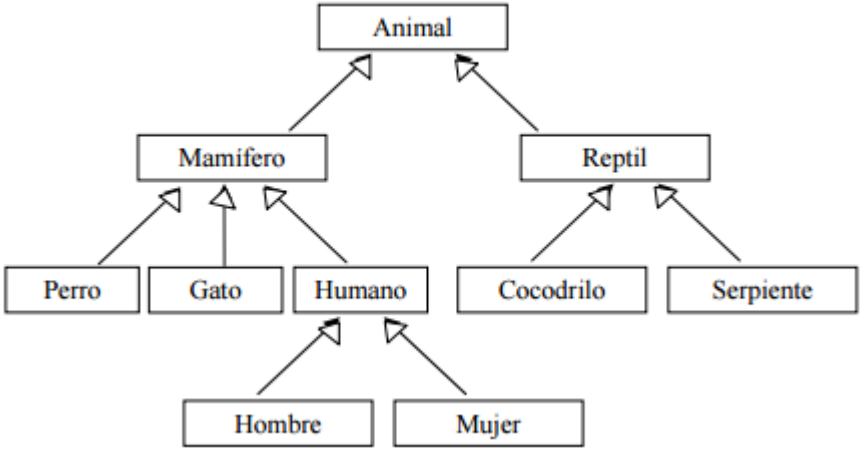
En la notación algorítmica:

Clase Punto3D Hereda de Punto2D

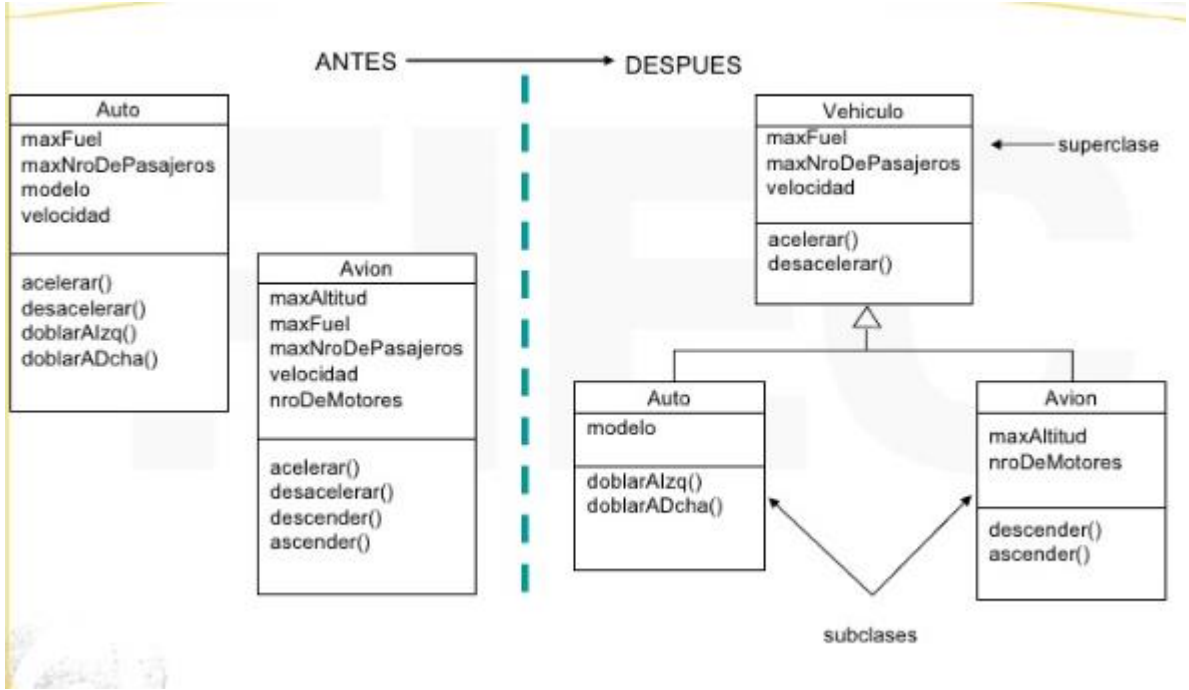
Relaciones entre clase

En el diagrama de clase la herencia se representa mediante una relación de generalización/ especificación, que se denota de la siguiente forma:

Relaciones entre clase



Relaciones entre clase



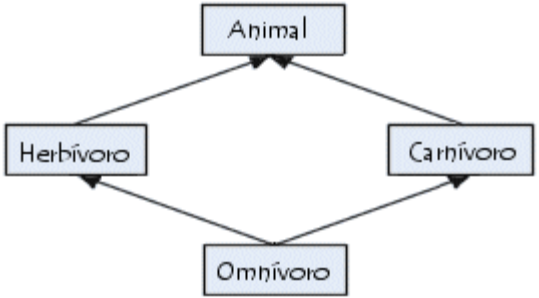
Relaciones entre clase

Tipos de Herencia:

Herencia simple: una clase (clase hija) puede heredar de otra clase (tener una clase padre).

Relaciones entre clase

Herencia múltiple: una clase (clase hija) puede heredar de otras clases padres (tener varias clases padres).



Relaciones entre clase

Ventajas de la herencia:

1) Evitar duplicidad y favorecer la reutilización de código. Las subclases utilizan el código de las superclases.
2) Facilitar el mantenimiento de aplicaciones. Podemos cambiar las clases que utilizamos fácilmente.
3) Facilitar la extensión de las aplicaciones. Podemos crear nuevas clases a partir de otras existentes.

Ejercicio

1) Realizar un ejemplo (Diseño) de herencia simple
2) Realizar un ejemplo (Diseño) de herencia múltiple

Relaciones entre clase

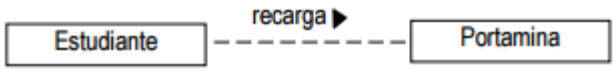
Asociación: Se establece cuando dos clases tienen una dependencia de utilización, es decir una clase utiliza atributos y/o métodos de otra para funcionar. Estas dos clases no necesariamente están en jerarquía, es decir, no necesariamente una clase es padre de la otra, a diferencia de las otras relaciones de clases.

Relaciones entre clase

Por ejemplo: Tenemos la clase humanos que utiliza la clase lápiz para escribir, o la clase tenedor para comer.

Para validar la asociación, la frase “usa un” debe tener sentido.

Su representación es:



Relaciones entre clase

Otro ejemplo:

¿Qué ejemplo se le ocurre de asociación?

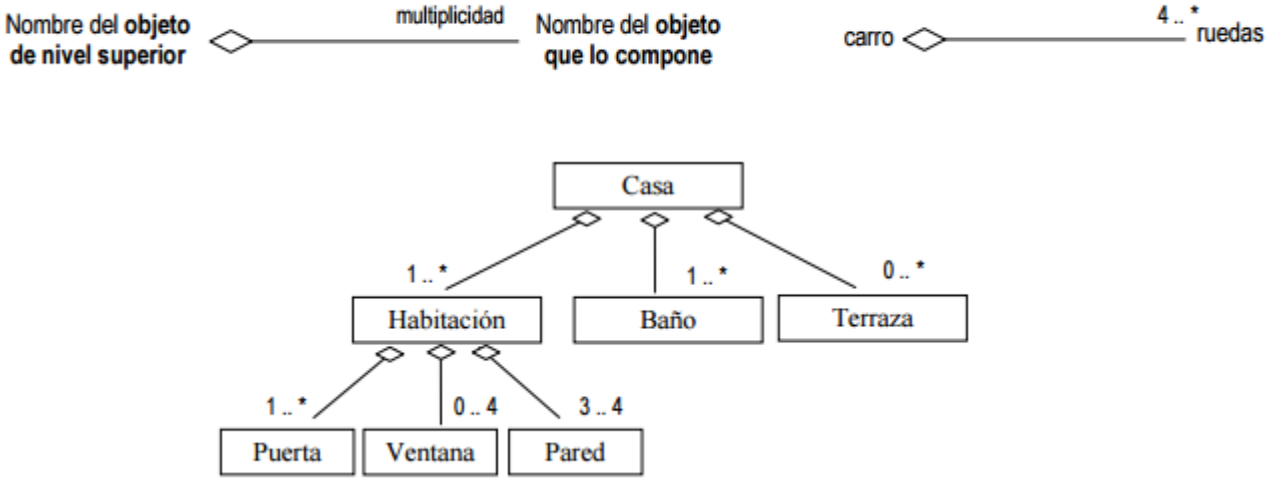
Relaciones entre clase

Agregación: las relaciones de agregación se basan en la idea de observar o entender un objeto como una composición de otros objetos.

Estas relaciones se conocen como “todo-parte”. El todo esta representado por la clase que aglutina a las otras clases.

Relaciones entre clase

Representación en el diagrama de clase:



Relaciones entre clase

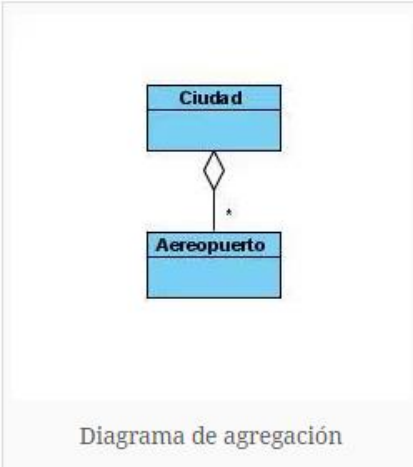


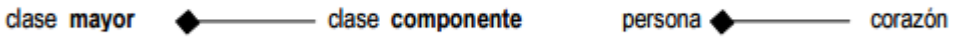
Diagrama de agregación

¿Qué ejemplo se le ocurre de agregación?

Relaciones entre clase

Composición: un componente es parte esencial de un elemento. La relación es más fuerte que el caso de agregación. Al punto que si el componente es eliminado o desaparece la mayor deja de existir.

Representación en diagrama de clases:



Relaciones entre clase

Ejemplo Composición:

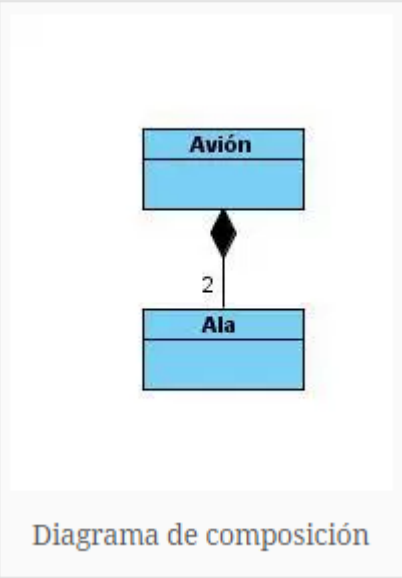


Diagrama de composición

El avión tiene sentido por sí solo, pero está compuesto por dos alas. Y estas serán siempre del mismo avión.

Ejercicio (Diseño UML)

Definir una clase que contenga cinco atributos y tres métodos que considere necesarios.

Ejercicio (Diseño UML)

Sobre la clase definida en el punto anterior, definir dos clases que deriven de esta.

Ejercicio (Diseño UML)

Pensemos como sería la clase Tren y la clase Impresora.

Ejercicio (Diseño UML)

Definir clase Rectángulo. Calcular su área y su perímetro.